On the Detection of Malware on Virtual Assistants Based on Behavioral Anomalies

Mahshid Noorani∗†, Spiros Mancoridis†, Steven Weber∗
∗Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA
†Department of Computer Science, Drexel University, Philadelphia, PA

Abstract—This work explores some of the security concerns pertaining to running software similar to Amazon Alexa home assistant on IoT-like platforms. We implement a behavioral-based malware detector and compare the effectiveness of different system attributes that are used in detecting malware, i.e., system calls, network traffic, and the integration of system call and network traffic features. Given the small number of malware samples for IoT devices, we create a parameterizable malware sample that mimics Alexa behavior to varying degrees, while exfiltrating data from the device to a remote host. The performance of our anomaly detector is evaluated based on how well it determines the presence of our parameterized malware on an Alexa-enabled IoT device.

I. INTRODUCTION

The Internet of Things (IoT) refers to the growing network of “smart objects.” The increase in popularity of IoT devices, due to their efficiency and convenience, has given rise to new security concerns. These devices reside on internal networks, have their own IP address, and allow communication with devices and systems that are connected to the Internet. Their ubiquity, large numbers, lack of cryptographic encryption, and weak default authentication make them highly attractive targets.

The interconnected nature of IoT devices means that every insecure device that is connected online has the potential to affect the security and resilience of the entire Internet. IoT devices are heterogeneous and specialized devices by nature. However, the variety and novelty of IoT devices has yet to yield a corpus of malware that is sufficiently large to employ machine learning algorithms. This makes anomaly detection methods for IoT device security more attractive, especially in the short term, until there are enough behavioral signatures for malware to train more sophisticated machine learning detection models for these devices.

Amazon Alexa Echo is an example of a popular IoT device. This intelligent virtual assistant processes natural language to assist people with tasks around their home. Alexa, similar to most IoT devices, is designed to accomplish simple and specialized tasks. We claim that this narrow spectrum of responsibilities, which is common to IoT devices in general, yields a higher chance of success using anomaly-based detection methods that identify deviations in measured statistics against a normal model of operation of Alexa.

In this work, we built a one-class Support Vector Machine (SVM) [1] for behavioral-based anomaly detection. The detector was separately trained on three sets of system attributes. We will describe how it is possible to detect the existence of malware on the Alexa device by first training an anomaly detector on operating system kernel layer system call data, then training an anomaly detector on network traffic data, and finally training an anomaly detector on a set of features integrated from system call and network features as shown on the left side of Figure 1.

These anomaly detectors are trained using only benign (normal) usage scenarios. Each detector is subsequently subjected to both regular usage scenarios and malware infections in order to demonstrate its effectiveness to distinguish normal Alexa operation from anomalous, perhaps malicious, operation in real time. While we demonstrate the effectiveness of our technique on Amazon Alexa, we anticipate that small variations of our technique can be used to create anomaly detectors for other IoT devices, because these devices are typically highly specialized and, hence, exhibit a predictable normal behavior. This is in contrast to creating an anomaly detector for a general purpose computer, which has a
more elaborate and varying behavioral profile.

To overcome the limitation of only having a small number of pre-existing malware samples for IoT devices, we decided to create a parameterizable malware sample that mimics Alexa behavior to various degrees, while exfiltrating data from the device to a remote host as shown in the right side of Figure 1. We tune the designed malware parameters to mimic Alexa behavior more closely based on the feedback received from our detector on which features are a better representation of Alexa behavior. The performance of each of our anomaly detectors is evaluated based on how well it determines the presence of our parameterized malware on an Alexa-enabled IoT device. In this work, we also have compiled a labeled dataset of Alexa-Pi system calls and network traffic features, which can be used in future studies of IoT devices.

II. PREVIOUS WORK

As IoT devices become more prevalent and are subjected to more security attacks, researchers can gather more malware samples and perform traditional malware detection. Meanwhile, these shortcomings have motivated the use of anomaly detection technique for exposing previously unseen threats. Behavioral-based anomaly detection does not require a prior knowledge of vulnerabilities for identifying previously unseen threats. In this method, a model of the system is created based on the normal behavior of the system and the deviation from the model yields an indication of compromise. Unlike general purpose computers, which consist of many different applications, IoT devices are very specialized. As a result, it is much easier to create an effective anomaly detector for malware detection on IoT devices. Others have argued that the novelty of IoT makes anomaly detection a superior choice even in the long term [2] [3] [4] [5].

In the prior art, data that was tainted by malware executions was used as part of the model training process. In contrast, this paper focuses on anomaly-based malware detection because of the limitations of traditional signature-based detectors (e.g., ineffective on zero-day attacks) and the fact that current IoT device do not yet have enough malware samples that can be used to build a robust model. Therefore, our focus is on developing detection techniques that do not require a priori information about specific threats, but rather are designed to detect unusual behavior from the IoT device.

Extensive research has been done on the effectiveness of a variety of system features for detecting system anomalies. System calls and network traffic are two categories of system features that are often used to detect system anomalies. The authors have done some related prior work where they developed techniques to detect anomalous behavior on home assistant devices such as Amazon Alexa Virtual Assistant IoT device [6]. Intelligent virtual assistants, such as Alexa-enabled IoT devices, typically have specialized functionality which makes their ordinary behavior easy to learn. The effectiveness of a semi-supervised anomaly detector, trained by ordinary system call sequences generated by the IoT device’s kernel, has been studied to identify malware activities without needing to have seen malware samples in training [6]. The results suggest that an anomaly detector based on system calls is capable of detecting the presence of malware samples with significantly high accuracy in a very short time, with only modest-sized training data.

Network behavioral modeling is another popular approach for malware detection and malware family classification [7]. Network traffic data represent the amount and type of traffic on a particular network. They contain information about sent and received communications on a device. The effectiveness of network traffic-based analysis for detecting anomalous behavior on a system has been tested using machine learning algorithms [8]. The network traffic data have been shown to be effective for training models that can identify anomalous behaviors on systems [9] [10].

Important advances have been made on malware detection in traditional personal computers over the last decade. However, adopting and adapting those techniques to smart devices is a challenging problem. For example, power consumption is one major constraint that makes it unaffordable to run traditional detection engines on IoT devices, while externalized (i.e., cloud-based) techniques raise many privacy concerns. A study of the cost and benefits of integrating system call and network traffic to perform anomaly based detection detection for IoT devices is, therefore, of great interest. IoT is a resource constrained environment, therefore we want to make sure we are doing the anomaly detection in a cost efficient manner. Each of these processes will have a cost in terms of energy, resources, and delay. In this research, we compare multiple detectors to learn if better results are obtained using only one of these detectors, or if a hybrid detector that uses both of these kinds of features is better.
III. ANOMALY DETECTION MODEL

To the best of our knowledge, currently, there are no malware samples, other than proof-of-concept ones, that target voice-controlled IoT devices that use Amazon Alexa. The lack of malicious training samples impedes researchers carrying out supervised malware detection techniques. Under the assumption that not enough malware samples are available, but will likely be available eventually, anomaly detection techniques are of great interest for protecting IoT devices in the short term. Additionally, the rapid variation and development of malware in general, namely as polymorphic and metamorphic malware [11], highlights a unique advantage of anomaly detection over supervised approaches.

This work employs one-class SVM for anomaly detection. This section begins by describing our data extraction and preprocessing methods in §III-A, and then proceeds with an overview of the anomaly detector.

A. Feature generation and preprocessing

Behavioral anomaly detection uses sequences of system calls and network traffic flows, and extracts features from them for the anomaly detector. Specifically, our detector relies on the traces of Linux OS kernel-level system calls that are generated by the processes running on the Amazon Alexa device and network traffic to and from Alexa-Pi.

These system calls describe what OS functions are used by each process executing on a computing platform. When a malware process starts executing, its system call traces will likely not match any of the benign traces that are currently executing or have executed in the past. System calls can be viewed as the language of machines, and feature extraction methods in natural language processing can be used to pre-process the system call traces.

Similar to prior work [6] we take a sequence of systems calls collected in an observation window of length $L$ as a data sample. Then, we employ the bag-of-$n$-grams model [12], [13] to produce the feature vector $x \in \mathbb{R}^p$, which is a vector of the number of the system call sequences that occurred in a small sliding window of length $L$ data samples.

Network traffic features are extracted from the processing of data captured in pcap files. Pcap files can be analyzed by using tools such as wireshark, tshark, or Python’s scipy package. In this work, we use the CICFlowMeter tool to extract features from the pcap files. The CICFlowMeter is an open source tool that generates bidirectional flows from pcap files, and extracts features from these flows. The selected features have shown to produce results with respect to both accuracy and performance, confirming that time-related features are conducive to producing good classifiers for encrypted traffic characterization [14].

In the collected network traffic data, each sent or received network packet counts as one observation. During the data collection in Malware mode, many packets are transferred between Alexa-Pi and Amazon servers as they would normally in Benign mode. Only a few packets contain information that is specific to malicious behavior. In other words, the malware class contains a combination of malware and benign behavior, all labeled as malicious. As a result, labeling all collected traffic in Malware mode as malicious activity results in an inseparable dataset. This behavior is demonstrated in the right side of Figure 2. We resolve this issue by grouping the dataset into $m$-second intervals. Each new batch contains traces of malicious behavior. As a result, our new malicious observations are more distinct form the benign traces as shown in the left side of Figure 2. The binning size is a tunable parameter which affects the malware detection accuracy of the model.

![Fig. 2: Malware and Benign Network Data w/ and w/o Grouping](image-url)

We use percentile aggregation to group the network traffic dataset. There is an undefined number of generated packets per $m$-seconds, since we do not know the exact number of samples per each time interval, our best bet is to capture a probability distribution estimation of the dataset using percentile aggregation. To create a reasonable-size feature vector we only aggregate the data by three,
which means replacing each feature column by three new columns representing the $25^{th}$, $50^{th}$ and $75^{th}$ percentile of the original feature. The new feature vector consists of 237 features.

237 features were extracted from network traffic data. Approximately 3300 features were extracted from system call data. This brings the total size of the feature set to approximately 3500 features. We use Principal Component Analysis (PCA) to reduce the dimensionality of our feature space. The number of PCs were decided on by calculating the cross-validated $r$-squared score for $n$ linear combinations of features. The cross-validated coefficient of determination is the proportion of the variance in the dependent variable that is predictable from the independent variable. We employ cross-validation on the data and take the number of PCs that corresponds to the maximum cross-validated $r$-squared score. Figure 3 shows the cross-validated $r$-squared score for each PC. There is always a peak in the graph that represent the optimum number of PCs. In this experiment, 20 is the selected number of PCs.

![Fig. 3: Significance of Principal Components in PCA](image)

**B. One-class SVM-based anomaly detection algorithm**

The one-class SVM finds a hyperplane that separates the training data from the origin with a margin that is as large as possible. The one-class SVM’s objective function minimizes the normalized weights vector $w$ of the hyperplane (which is equivalent to maximizing the margin), and the objective function is also penalized by points that lie on the wrong side of the hyperplane (i.e., the side wherein the origin lies). The test statistic of the one-class SVM is the distance to the separating hyperplane: $y = w^T \phi(x) - \rho$ [15].

**IV. EXPERIMENTAL SETUP**

In this work, we implement a real IoT system almost identical to having an Amazon Echo personal assistant at home. We then design a behavioral based malware detector to detect malware by measuring the network traffic statistics between Amazon servers to the Echo device and back. In order to collect kernel level data and network traffic statistics, we need an open platform. Amazon Echo is a closed system, which limits our access level to this device. We built an Alexa-Pi IoT device, which has all of the capabilities of Amazon Echo and allows for data collection by elevating our system privileges. In our experiments, we used a Raspberry Pi 2 Model B running Alexa Voice Service in place of an Amazon Echo Dot, which is one of the most popular smart speakers.

**A. Raspberry-Pi setup**

The model of the device running the Alexa Voice Service is a Raspberry Pi 2 Model B. The operating system installed on the device is a Raspbian-Stretch as provided by the Raspberry Pi foundation website. We use this device as a surrogate for the Amazon Echo Dot (Gen 2) for two reasons.

First, the Amazon Echo Dot runs Amazon’s Fire OS. Because Fire OS is proprietary, we are unable to do many of the things required to perform our data collection on the device. However, the Raspberry Pi runs the Raspbian-Stretch operating system, which is open source and allows us to customize the device and access the OS kernel. While the two operating systems are different from one another, they are both Unix-based, and as such, have a similar low-level functionality.

Second, the Raspberry Pi 2 Model B’s central processing unit is the Broadcom BCM2836 [16], while the Amazon Echo Dot’s central processing unit is the MediaTek MT8163V [17], both of which are quad-core processors whose underlying architecture is the ARM Cortex-A53 micro-architecture, implementing the ARMv8-A 64-bit instruction set [16], [18].

**B. Alexa installation**

The Alexa-Enabled devices available from Amazon (such as the Echo) are not suitable for collecting the data we use as features for our detection model. As such, we installed the Amazon Voice Service on a Raspberry Pi 2 Model B. To achieve feature parity with an official Alexa-Enabled device, one must register as a developer with Amazon, register the device upon which the Alexa Voice Service will be installed, and download some libraries, programs, and security certificates from Amazon. With the addition of a USB microphone and analog speakers connected via the Raspberry Pi’s 3.5mm audio
output port, a user can query the device in the same way as one would with a device such as an Amazon Echo and receive the same results.

C. Exercising the Alexa capabilities

Currently, there is no interface available to the public to facilitate the automatic interaction with the Alexa Voice Service for practicing built-in capabilities. The majority of computation performed when making queries is handled at a remote server rather than the Alexa-Enabled device itself. Consequently, the system calls made on the device itself when handling different queries are very similar, as the device mostly establishes a connection to Amazon’s servers and offloads the query. Furthermore, in practical use, the device is not handling queries at all, but rather listening for a wake word. During this passive listening state, the Amazon Voice Service is still running and awaiting input. This means that even when the device is not actively handling a query, it still continually makes system calls as part of the service. As such, the pattern of behavior demonstrated while the device is idle is still useful in building a realistic model for anomaly detection. When gathering data for our experiments, we exercised a broad set of Alexa built-in capabilities from books, calendar, weather, music, and standard built-in skills categories. We also gathered data when Alexa was idle, both in a quiet environment as well as in an environment with ambient noise.

D. Malware Design

Alexa-Pi creates a recording of the user’s speech after hearing the Wake-up word. These recordings are sent to the Amazon servers for further analysis. The Amazon server then sends an mp3 file back to Alexa-Pi as a response to the query, which gets saved in a temporary folder on the Alexa-Pi device. The recordings remain in the temp file until the next time Alexa hears the Wake word. Alexa-Pi also sends routine token update (a token that indicates the Alexa-Pi is up and authorized to communicate with Amazon servers) requests to the Amazon servers approximately every 3570 seconds. This reoccurring behavior allows us to schedule the malware to synchronize the sending of the recording with the Alexa token update request time. This results in more stealth and less detectable malicious behavior in Idle mode. However, sending the zip files to a remote server only every 3570 seconds, while Alexa generates many recordings in Query mode, requires us to save the zipped recordings in a folder and send them all in one attempt at 3570-second intervals. Since the number of packets and the packet size would not match Alexa’s behavior, this approach would result in an easily detectable malware in Query mode. Hence, we used a second approach with the intent for our malware to remain more stealth in Query mode.

The first generation of malware mimics Alexa’s behavior by exfiltrating the recordings saved by Alexa. We generated a family of malware that varies based on the intensity of this operation. For example, we collected network and system call data while zipping and sending Alexa recordings to a remote server every $m$ seconds. We then varied the value of $m$ to generate a family of malware, which vary in intensity of operation. We then create three malware samples by setting the value of $m$ to 5, 10, and 30 seconds. Since we query Alexa continuously in the query mode, Alexa sends many packets to Amazon servers, which create a perfect opportunity for us to send the malicious packets in disguise. The malware detectability decreases as the intensity of operation decreases. Hence, the malware with the intensity of operation set to 30 seconds is the most successful in deceiving the detector.

The second generation of malware collects feedback from our designed detector and learns which features used in our detection mechanism is most likely to trick the detector. The selected feature is set as a new parameter of our malware and adds an additional degree of freedom to our parameterized malware. The selected parameter in this work is the Flow Duration feature. To ensure the validity of the chosen parameter for the 2nd generation of our malware, we tested the malware’s detectability by comparing the performance of the detector in two scenarios. In scenario I, we enforced no constraints on the range of the flow duration parameter’s values. In scenario II, we set the malware’s flow duration value to a constant number that mimics the average of Alexa-Pi’s normal traffic’s flow duration. The comparison of the detectibility of the malware in scenarios I and II confirmed that the malware in scenario II is more successful in deceiving the detector.

V. EXPERIMENTAL RESULTS AND COMPARISONS

We have conducted our experiment in three modes, namely Idle, Ambient, and Query. Idle mode is when system call and network traffic data is collected in an environment with no background noise and no ambient user conversations. Ambient mode is when the data is collected in an environment where there is an active conversation happening in the room but the Wake-up word, “Alexa”, is not used in the conversation. Query
mode is when the user actively uses the Wake-up word to initiate an interaction with the Alexa-Pi device. In each mode, we can create six scenarios by tuning the parameters of our malware. The performance of our three anomaly detectors in different modes are compared in Figures 5, 6, 7.

The two selected scenarios in the left side of Figure 4 are the scenarios in which the malware is most and least likely to trick the detector. The AUC for each detector is summarized in a separate table: Tables I, II, and III. Each table reports the Area Under the ROC Curve (AUC) for the two selected malware modes in each of the three environments (Idle, Ambient, and Query). The comparison of AUC in each environment suggests that a combined network traffic and system call detector performs better than the network traffic or system call detectors individually. We observe that our designed malware is more detectable based on the network traffic analysis. The system call detector does not show a high detection accuracy for our malware on its own, however it seems to detect the anomalies based on characteristics that are overlooked by the network traffic detector. Hence, the combined network traffic and system call detector consistently reports higher detection.

Fig. 4: The Selected Malware Modes Shown in ROC Curves (right) and Tables (left)

### TABLE I: System Call Detector AUC Values For Each Mode

<table>
<thead>
<tr>
<th>Mode</th>
<th>Idle</th>
<th>Ambient</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non Alexa-like Flow Duration</td>
<td>1.00</td>
<td>0.99</td>
<td>0.98</td>
</tr>
<tr>
<td>Alexa-like Flow Duration</td>
<td>1.00</td>
<td>0.98</td>
<td>0.76</td>
</tr>
</tbody>
</table>

### TABLE II: Network Traffic Detector AUC Values For Each Mode

<table>
<thead>
<tr>
<th>Mode</th>
<th>Idle</th>
<th>Ambient</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non Alexa-like Flow Duration</td>
<td>1.00</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>Alexa-like Flow Duration</td>
<td>0.87</td>
<td>0.94</td>
<td>0.73</td>
</tr>
</tbody>
</table>

### TABLE III: Combined System Call and Network Traffic Detector AUC Values For Each Mode

<table>
<thead>
<tr>
<th>Mode</th>
<th>Idle</th>
<th>Ambient</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non Alexa-like Flow Duration</td>
<td>1.00</td>
<td>1.00</td>
<td>0.99</td>
</tr>
<tr>
<td>Alexa-like Flow Duration</td>
<td>1.00</td>
<td>0.99</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Figures 5, 6, 7 show the accuracy of our model for the scenario in which the malware is tuned to behave the most similar to an Alexa-Pi IoT device as shown in the right side of Figure 4. The malware is detectable with almost a 100 percent accuracy in Idle mode. The detection rate declines to 99 percent in Ambient mode, and goes down to a 89 percent in the Query mode. This decline in accuracy is expected as our malware is designed to mimic Alexa-Pi in Query mode. One possible reason for the low detection accuracy using the system call detector is that our custom-made malware does not perform traditional malicious activities, such as malware replication and propagation, which would make it sufficiently different from Alexa-Pi.

There is a discrepancy between the system calls observed when the Alexa-Pi device is Idle and placed in a quiet room, and those observed when the malware-free device is placed in a noisy room (i.e., Ambient) or when the malware-free device is being queried. The system calls in the Ambient and Query modes are more similar to each other than to the system calls in the Idle and Ambient modes. This suggests that despite the presence, or lack of presence, of the Wake word in a conversation, the Alexa-Pi device behaves in a similar manner. This might be a result of the way the Alexa API client operates. In both Ambient and Query mode, the Alexa API client analyzes the conversations in the room and sends them to the Wake Word Detection Engine. The Detection Engine determines if the audio file requires a response. Based on our research, all audio files, with or without the wake word, are sent to Amazon servers by default. The Alexa app allows access to these files and gives the user permission to erase the saved recordings.

VI. CONCLUSIONS

This work focused on anomaly-based malware detection because of the limitations of traditional signature-based detectors (e.g., ineffective on zero-day attacks) and the fact that current IoT devices do not yet have enough malware samples that can be used to build an accurate malware detection model. Therefore, our focus was on developing detection techniques that do not require a prior knowledge of specific threats, but rather are designed to detect notable deviations from normal Alexa’s behavior.

We designed a real IoT system and collected system call and network traffic data. We described the creation of behavioral anomaly detection systems designed to detect the execution of malware on an Amazon Alexa IoT device. Three detectors were designed based on different system features, i.e., network traffic, system call, and an integration of system call and network traffic features. We then created a parameterizable malware that mimics Alexa’s behavior to varying degrees.
The key findings and contributions in this work are as follows: (I) We observed that a detector based on combined system call and network traffic features provides better detection compared to system call-based or network traffic-based detectors individually. (II) This work focused on the co-evolution of data exfiltration malware targeting IoT devices and anomaly detectors based on network and system call data. The focus of this work was not to optimize the detector or the malware, in general, but, rather, to evolve the malware based on the input from the detector, and vice versa. (III) We created a dataset consisting of network traffic and system call data to study IoT devices. We are planning to make this dataset available on GitHub.

In this work, we studied the cost and benefits of combining system call and network traffic features of an anomaly-based detector for IoT devices. However, we did not determine the optimal contribution of each specific detector to the combined detector.

ACKNOWLEDGMENT

The work was funded in part by Spiros Mancoridis’ Auerbach Berger Chair in Cybersecurity.
REFERENCES


